

Rony G. Flatscher

# Introduction to REXX and ooRexx

From REXX to Open Object Rexx (ooRexx)

---

Edition: 1.01, 2013-11-03 (Version 101.20131103)

Copyright © 2013 Rony G. Flatscher

<http://www.RonyRexx.net>

c/o WU Wien, Augasse 2-6, A-1090 Wien

All rights reserved.

## Acknowledgements

The author thanks the following persons for their feedback, proof reading and help in creating free REXX-related art (in alphabetical order):

Gilbert *Barmwater* (U.S.A.): feedback, proof reading

Daniel A. *Flatscher* (Austria): proof reading

Howard *Fosdick* (U.S.A.): feedback, proof reading

René Vincent *Jansen* (The Netherlands): feedback, proof reading

Les *Koehler* (U.S.A.): feedback, proof reading

Gerald *Leitner* (Austria): feedback, proof reading

DI Walter *Pachl* (Austria): feedback, proof reading

Graham *Wilson* (South Africa): art including icons for BSF4ooREXX

Jon "Sahananda" *Wolfers* (United Kingdom): feedback, proof reading

## Foreword

**A Brief History of the Rexx programming language.** In 1979 *Mike F. Cowlshaw* (MFC), an English gentleman working for IBM, devised a “human centric” programming language for the IBM mainframes that was easier to understand and to program than the arcane mainframe batch language named *Exec 2*. The design work was carried out at the IBM Research facilities in Hursley under the management of *Dr. Brian Marks*. It was probably the first time in the history of programming language design that IBMers interconnected worldwide via the IBM internal network were able to influence the design by studying the distributed specifications and giving feedback, like *Les Koehler* from IBM USA.

IBM later defined the *REXX* programming language to be the strategic batch/scripting language on all of its operating systems via IBM's *SAA* (System Application Architecture) standard. Another outstanding IBM employee who has probably been the only person to create Rexx interpreters multiple times for multiple operating systems is *Rick McGuire*, who led the development and maintenance of the *IBM SAA REXX* interpreters.

The IBM lab in Vienna created a Rexx compiler for its mainframe *REXX* (*Klaus Hansjakob, Walter Pachtl*), which is being sold and maintained by IBM to this very day.

*Mike F. Cowlshaw* documented the *REXX* language in a book named “The Rexx Language” also known as “TRL” and he later became one of the few IBM Fellows<sup>1</sup> due to his continuing innovative and influential work (he is also attributed to be the person who made Java a strategic language and platform within IBM, having ported Java to the IBM OS/2 PC operating system in the 90'ies).

The high impact of the Rexx language can be witnessed by the appearance of numerous non-IBM implementations of the Rexx language, e.g. *Regina* (open source, *Anders Christensen, Mark Hessling*), *Rexx/imc* (open source, *Ian M. Collier*) or *BREXX* (open source, *Vassilis N. Vlachoudis*), but also proprietary and commercial Rexx interpreters like *ARexx* (part of the Amiga operating system), Novell Netware's Rexx (in the 90'ies), Workstation Unix

---

<sup>1</sup> An IBM Fellow is free to research and to work, very much like professors at Universities, who have the freedom to freely determine what they research and what they teach.

## Foreword

Rexx. Some pointers to various Rexx interpreters can be found at <http://www.rexxla.org/rexxlang/mfc/rexxplat.html>.

**ANSI/INCITS REXX Standard.** In 1996 the American National Standards Institute (ANSI) working group X3J18 finalized the “American National Standard for Information Systems – Programming Language REXX”. After ANSI got renamed to “INCITS (InterNational Committee for Information Technology Standards, <http://www.incits.org/>)” the respective standard was named “INCITS 274 :1996 [R2001]”. In 2007 the INCITS 274 REXX standard was extended for another period of ten years, reflecting the importance of the Rexx language in the industry.

The INCITS 274 REXX standards on decimal arithmetic served as the basis for defining decimal formats in IEE 754-2008 and ISO/IEC/IEEE 60559:2011. An ANSI C implementation has been created by *Mike F. Cowlshaw* (<http://www.speleotrove.com/mfc/>), who on behalf of IBM has been a driving force behind standardizing decimal arithmetics in the context of IEEE, Java JSR-13 and who also implemented an open source `decNumber` package in ANSI C.

**A Brief History of the ooRexx Language.** At the end of the 80'ies Dr. Brian Marks († 2012) oversaw another interesting project, named “Oryx” with the technical lead of *Simon Nash*. The aim of this project was to experiment with a Rexx interpreter that extends the Rexx language with object-oriented features. This work would later lead, under the auspices of *Rick McGuire*, to the IBM product “Object REXX”. It was first distributed with OS/2 Warp 4 in 1997, versions for IBM AIX and MS Windows were created and sold as well by IBM.

In 2004 IBM handed the source code of “Object REXX” over to the non-profit special interest group (SIG) “Rexx Language Association (RexxLA, <http://www.RexxLA.org/>)”. RexxLA published the first open source version of IBM's “Object REXX” as “Open Object Rexx 3.0 (<http://www.ooRexx.org/>)” in 2005. The lead architect of this now open-source project has been *Rick McGuire*, who has been working in his own time on the open source version of ooRexx ever since.

### What Is ooRexx?

- A “classic Rexx” interpreter. ooRexx runs any “classic Rexx” program and can be used to write “classic Rexx” programs. There is no need to use any of its new features that extend the Rexx language.

- *An object-oriented Rexx language, hence “ooRexx”*: ooRexx comes with many useful classes (data types) and offers state-of-the art object-oriented features, devised in a “human-centric” way. Among other great things, ooRexx makes it easy to create multithreaded Rexx programs!
- *Fast and powerful*: ooRexx is a fast Rexx interpreter. It is a very powerful interpreter, which can be invoked from C++ or Java (via BSF4ooRexx) to allow Rexx and ooRexx macros/programs to run with C++ and Java applications. For C++ and Java applications it is possible to run multiple ooRexx interpreter instances in the same process space, each of which may execute even multithreaded Rexx code!
- *Great documentation*: IBM not only donated the source code for open sourcing to RexxLA, but also the excellent and professional technical documentation, which has been kept up-to-date. All the ooRexx documentation is available in the form of HTML and PDF-files, which can be nicely printed as books. The documentation is also directly available via the Internet: <http://www.oorexx.org/docs/>.
- *Free and open source*: originally created by IBM and marketed as “Object REXX”, RexxLA received the sources for publishing, maintaining, and enhancing this powerful Rexx interpreter. RexxLA distributes ooRexx with source code for free: <http://www.ooRexx.org>.
- *Multiplatform*: ooRexx is available in 32 and 64 bit versions for the operating systems AIX, Linux, MacOSX, Windows, and can be built for any Unix implementation. Rexx programs written in one operating system environment can execute in any other operating system environment.
- *Extensible*: ooRexx comes with a powerful and easy to use C++ API which is documented in one of the accompanying ooRexx documentation PDFs (cf. rexxpg.pdf). This allows you to extend ooRexx with functions and methods implemented in C++, but also to bridge Rexx with other programming infrastructures like Java (cf. “BSF4ooRexx”). In addition it allows C++ applications to create Rexx interpreter instances which execute Rexx programs. This way it is fairly easy/simple to employ ooRexx as a macro language for any C++

## Foreword

applications. The BSF4ooRexx extension package provides the same functionality for Java applications.

The author wishes to acknowledge the following persons important to the Rexx world in the context of RexxLA in alphabetic order: *Gil Barmwater* (vice president), *Mike F. Cowlshaw* (honorary board member), *Chip Davis* (past president), *Mark Hessling* (board member), *René Vincent Jansen* (current president), *Les Koehler* (secretary/treasurer), *Lee Peedin* (past president), *Pam Taylor* (board member), *Jon “Sahananda” Wolfers* (board member).

Of course all the developers of ooRexx (including past) are acknowledged hereby (in alphabetic order): *David Ashley*, *Jean-Louis Faucher*, *Mark Hessling*, *Moritz Hoffmann*, *Rick McGuire*, *Mark Miesfeld*, *Lee Peedin*, *David Ruggles*, *Bruce Skelly*, *Rainer Tammer*, *Jon Wolfers*.

## About this Book

This book introduces the programming language `Open Object Rexx`, also known as "`ooRexx`" in two steps:

1. Chapter 1 'The Rexx Language ("Everything Is a String")' introduces the `Rexx` programming language that was created in 1979 by the IBM employee Mike F. Cowlshaw who later became an IBM Fellow due to his work on `Rexx`. The most important design philosophy for the language was the principle of "human-orientation", making it easy for programmers to create programs in the `Rexx` language compared to the arcane IBM mainframe batch language `Exec 2` which `Rexx` successfully replaced. One key success factor of the `Rexx` language has been its easy English-like syntax that makes it easy to learn, fast to comprehend, easy to apply and inexpensive to maintain. `Rexx` programs can be read almost like prose. As `ooRexx` is backwardly compatible to `Rexx` it can be used to learn `Rexx` and thereby the fundamentals of programming. The concepts in this chapter apply generally to all existing `Rexx` interpreters, which sometimes are called "classic `Rexx`" interpreters (as opposed to `ooRexx`, which is a leading edge `Rexx` interpreter that extends classic `Rexx` nicely into the object-oriented world). `ooRexx`-only features are highlighted in the text.
2. Chapter 2 'Extensions to the Rexx Language by `ooRexx`' documents the `ooRexx`-only keyword instructions (`LOOP`, `RAISE`, `USE`) and enhancements to the `Rexx` language like short hand assignment operators (e.g. `+=`) and the directives `::routine` and `::requires` that may prove quite helpful to "classic `Rexx`" programmers.
3. Chapter 3 'The `ooRexx` Language ("Everything Is an Object")' builds upon the previous chapters and introduces the fundamental concepts of what is known as the "object-oriented (OO) paradigm". This is followed by an overview of the numerous new classes (data types) that come with `ooRexx` and which could be exploited by `Rexx` programmers to ease their programming life considerably in most cases. At the end of this chapter the reader should understand the OO-concepts and be able to take advantage of these new, powerful features!
4. Chapter '4 Reaching Out with `ooRexx`' opens with useful information about the `ooRexx` runtime system, followed by a categorized overview

## About this Book

of the ooRexx classes (data types, types) that are installed with the interpreter. The ooRexx programmer can directly use these ooRexx classes and take advantage of the features they implement.

5. Chapter 5 'Advanced Topics' introduces the interested reader to defining and implementing Rexx classes (data types), which is very easy and straight-forward. For those programmers who need the ability to create Rexx programs in which parts are executed concurrently, there is a concluding section which explains and demonstrates how easy it is to do that with ooRexx.

The structure and contents of the book are aimed at people who are interested in learning programming in Rexx and afterwards ooRexx. Still, it aims to introduce and demonstrate the concepts in a very concise, yet understandable manner. The reader is advised to consult the excellent ooRexx reference documentation, which completely documents ooRexx and is available as a nicely formatted PDF-book, named `rexxref.pdf` (<http://www.ooress.org/docs/rexxref/rexxref.pdf>).

The way this book is written should also allow professional programmers to skim the book and learn about the fundamentals of Rexx and ooRexx by looking out for the definition boxes that are formatted like this:

This is how a definition box is formatted. Definition boxes allow you to quickly get (re-)acquainted with the fundamental concepts that are taught in a chapter. This book will sometimes directly use the definitions of the ANSI/INCITS Rexx standard if possible in this book's context.

In addition, numerous little "nutshell programs" or "code snippets" demonstrate how to apply the introduced concepts. These programs, as short as they may seem, are full programs that can be executed as is by the ooRexx interpreter, yielding the output that is sometimes depicted alongside the program as well. "Nutshell programs" are formatted like this:

```
say "Hello world, this is Rexx speaking"
```

The above program will output the string *Hello world, this is Rexx speaking*.

Alternatively, ooRexx for Windows comes with a GUI program (menu entry named "Try Rexx (GUI)") which allows you to enter Rexx code and execute it with the push of a button. ooRexx users on Linux or MacOSX might want to



install BSF4ooRexx<sup>2</sup> (<https://sourceforge.net/projects/bsf4oorexx/files/GA/>) which comes with a comparable GUI program (menu entry named “GUI RexxTry Program (ooRexxTry.rxj)”).

Finally, ooRexx can be downloaded for free from one of the following locations:

- <http://www.oorexx.org/download.html>
- <https://sourceforge.net/projects/oorexx/files/>

There are editors that support Rexx syntax highlighting, for example the following two free and open source editors:

- “The Hessling Editor (THE)”, which uses Rexx as its macro language, URL: <http://hessling-editor.sourceforge.net/>
- “vim (vi improved)”, a part of many Linux distributions, is generally available for all operating systems, URL: <http://www.vim.org/>

---

<sup>2</sup> This GUI program is also available for Windows, if “BSF4ooRexx” gets installed there. “BSF4ooRexx” is an ooRexx external function package that allows ooRexx programs to interact directly with Java, which gets camouflaged as ooRexx. At the time of writing this external function package is available for Linux, MacOSX and Windows. Cf. 4.1 Exploiting Java on All Platforms, p. 157 below.



## Table of Contents

Acknowledgements .....	2
Foreword .....	iii
About this Book .....	vii
Table of Contents .....	xi
List of Figures .....	xv
List of Tables .....	xv
List of Codes (Nutshell Examples) .....	xv
Part I .....	21
1 The Rexx Language (“Everything Is a String”) .....	23
1.1 “Hello world!” in Rexx .....	23
1.2 Fundamental Language Concepts .....	24
1.3 Building Blocks and Definitions .....	26
1.3.1 Characters Allowed in a Rexx Program .....	26
1.3.2 Comments .....	27
1.3.3 Literal Strings .....	28
1.3.4 String Values .....	29
1.3.5 Symbols (Names) .....	30
1.3.6 Environment Symbols (ooRexx) .....	31
1.3.7 Operators .....	31
1.3.8 Expressions .....	34
1.3.8.1 Functions .....	34
1.3.8.2 String Concatenation Expressions .....	38
1.3.8.3 Arithmetic Expressions .....	39
1.3.8.4 Boolean Expressions .....	41
1.3.8.4.1 Comparisons .....	41
Comparing String Values .....	41
Comparing Numbers .....	42
1.3.8.4.2 Negator .....	43
1.3.8.4.3 Combining Boolean Values .....	43
1.3.9 Clauses .....	45
1.3.10 Normalizing Clauses (“Behind the Curtain”) .....	46
1.4 Writing Rexx Programs .....	49
1.4.1 Assignment Instructions (Variables) .....	49
1.4.1.1 Shorthand Assignment Instructions (ooRexx) .....	50
1.4.1.2 Stem Variables .....	51
1.4.1.3 Variable Names that Rexx May Use Without Notice .....	52
1.4.2 Label Instructions (CALL, SIGNAL, EXIT, RETURN, PROCEDURE, EXPOSE Keyword Instructions) .....	53
1.4.3 Message Instructions (ooRexx) .....	59
1.4.4 Keyword Instructions .....	61
1.4.4.1 SAY (Output of Strings) .....	64
1.4.4.2 IF (Choose) .....	64
1.4.4.3 SELECT (Alternatives), LEAVE (ooRexx) .....	65
1.4.4.4 DO (Block), ITERATE, LEAVE .....	66
1.4.4.5 LOOP (ooRexx), ITERATE, LEAVE .....	72

## Table of Contents

1.4.4.6 The Rexx QUEUE (PUSH, QUEUE, PARSE PULL, PULL) ....	72
1.4.4.7 PARSE .....	74
1.4.4.7.1 Parsing Using a Blank Delimited Template .....	74
1.4.4.7.2 Parsing Using Literal Strings as Delimiter .....	76
1.4.4.7.3 Parsing Using Positions and Lengths .....	77
1.4.4.7.4 PARSE VALUE .....	80
1.4.4.7.5 PARSE ARG, ARG .....	81
1.4.4.7.6 PARSE PULL, PULL .....	82
1.4.4.7.7 PARSE SOURCE .....	82
1.4.4.7.8 PARSE VERSION .....	83
1.4.4.8 TRACE .....	83
1.4.4.9 INTERPRET .....	84
1.4.5 Command Instructions, ADDRESS .....	85
2 Extensions to the Rexx Language by ooRexx .....	87
2.1 USE ARG .....	88
2.2 Trapping and Raising Conditions (SIGNAL CALL ON OFF, RAISE) .	89
2.3 Directives .....	93
2.3.1 The ::ROUTINE Directive .....	93
2.3.2 The ::REQUIRES Directive .....	98
2.4 The Big Picture .....	101
Part II .....	103
3 The ooRexx Language (“Everything Is an Object”) .....	105
3.1 Running ooRexx Programs .....	106
3.1.1 Runtime Environment .....	107
3.1.2 Objects (Values, Instances), Classes (Data Types) .....	109
3.1.2.1 Messages: Interacting with Objects .....	109
3.1.2.2 Classes: Attributes, Methods .....	110
3.1.2.3 Classes (Data Types) Organized as a Class Hierarchy .....	112
3.1.2.4 Unknown Messages .....	114
3.2 The ooRexx Built-in Classes .....	115
3.2.1 Categorizing the ooRexx Classes .....	115
3.2.2 The Fundamental ooRexx Classes .....	115
3.2.2.1 The .Object Class .....	116
3.2.2.2 The .Class Class (the ooRexx Metaclass) .....	117
3.2.2.3 The .Method Class .....	119
3.2.2.4 The .Message Class .....	120
3.2.2.5 The .Routine Class .....	121
3.2.2.6 The .Package Class .....	122
3.2.3 The Classic Rexx Classes .....	123
3.2.3.1 The .String Class .....	123
3.2.3.2 The .Stem Class .....	125
3.2.3.3 The .Stream Class .....	127
3.2.4 The ooRexx Collection Classes .....	128
3.2.4.1 The .OrderedCollection Classes .....	130
3.2.4.1.1 The .Array Class .....	130
3.2.4.1.2 The .List Class .....	133
3.2.4.1.3 The .Queue Class .....	134

3.2.4.1.4 The .CircularQueue Class .....	135
3.2.4.2 The .MapCollection Classes .....	136
3.2.4.2.1 The .Directory Class .....	137
3.2.4.2.2 The .Relation Class .....	140
3.2.4.2.3 The .Table Class .....	142
3.2.4.3 The .SetCollection Classes .....	142
3.2.4.3.1 The .Bag Class .....	143
3.2.4.3.2 The .Set Class .....	144
3.2.4.4 Setlike Operations on Collections .....	145
3.2.4.4.1 Example: Setlike Operations with .Bag and .Bag .....	146
3.2.4.4.2 Example: Setlike Operations with .Set and .Bag .....	147
3.2.4.4.3 Example: Setlike Operations with .Bag and .Set .....	148
3.2.5 Miscellaneous ooRexx Classes .....	149
3.2.5.1 The .Alarm Class .....	149
3.2.5.2 The .Comparator Classes .....	150
3.2.5.3 The .DateTime and .TimeSpan Classes .....	151
3.2.5.4 The .File Class .....	153
3.2.5.5 The .Monitor Class .....	154
3.2.5.6 The .MutableBuffer Class .....	155
3.2.5.7 The .RexxContext Class .....	156
4 Reaching Out with ooRexx .....	157
4.1 Exploiting Java on All Platforms .....	157
4.1.1 A Brief Overview of BSF4ooRexx .....	158
4.1.1.1 BSF4ooRexx Menu .....	158
4.1.1.2 ooRexxTry.rxj - A Platform Independent GUI for ooRexx ..	159
4.1.2 The ooRexx Package BSF.CLS .....	160
4.1.3 The ooRexx Package UNO.CLS .....	162
4.1.4 Further Information .....	166
4.2 Windows Platform Only .....	168
4.2.1 A Brief Overview of COM, OLE, ActiveX .....	169
4.2.2 The .OLEObject Class .....	170
4.2.2.1 Rosetta Stone: Visual Basic to/from ooRexx .....	174
4.2.2.2 Some Further Information .....	179
4.2.3 The ooDialog Framework .....	180
4.2.4 Additional ooRexx Windows Classes .....	181
5 Advanced Topics .....	183
5.1 A Few Things that Might Be Helpful to Know .....	183
5.1.1 About ooRexx Directives .....	183
5.1.2 About ooRexx Scopes .....	184
5.1.3 About the .methods Environment Symbol .....	185
5.1.4 About Cascading Messages .....	186
5.1.5 About Required String Values .....	187
5.1.6 About Special ooRexx Methods .....	188
5.1.6.1 Method "init" (Constructor) .....	189
5.1.6.2 Method "unInit" (Destructor) .....	189
5.1.6.3 Method "unknown" .....	191
5.1.6.4 Method "string" .....	191

## Table of Contents

5.1.6.5 Method “makeString” .....	192
5.1.6.6 Method “makeArray” .....	194
5.1.6.7 Comparison Methods .....	194
5.2 Defining ooRexx Classes .....	201
5.2.1 Abstract Data Type (ADT) .....	201
5.2.2 Implementing an ADT with Directives (::CLASS, ::METHOD, ::ATTRIBUTE, ::CONSTANT Directives) .....	203
5.2.2.1 The ::CLASS Directive .....	203
5.2.2.2 The ::ATTRIBUTE and the ::METHOD Directives .....	205
5.2.2.2.1 Method Scope (EXPOSE) .....	210
5.2.2.2.2 “Self” and “Super” in Method Routines .....	210
5.2.2.2.3 FORWARD .....	212
5.2.2.3 The ::CONSTANT Directive .....	212
5.2.3 Examples .....	213
5.2.3.1 Creating a Hierarchy of Classes .....	213
5.2.3.2 Employing Multiple Inheritance .....	215
5.3 Multithreaded Programming .....	217
5.3.1 REPLY, .Alarm, .Object and .Message .....	218
5.3.1.1 REPLY .....	218
5.3.1.2 The .Alarm Class .....	220
5.3.1.3 Method “start” of the .Object Class .....	221
5.3.1.4 Method “start” of the .Message Class .....	221
5.3.2 Synchronizing Rexx Threads (GUARD) .....	222
5.3.2.1 Synchronizing Concurrently Running Method Routines ...	223
5.3.2.2 Waiter .....	226
5.3.2.3 Producer and Consumer .....	228
Index .....	233
Some oo Rexx-Related World-Wide-Web Links .....	ccliv

## List of Figures

Figure 1: .bsfDialog's messageBox on Linux, MacOSX and Windows. ....	162
Figure 2: ooRexxTry.rxj:a Portable GUI for Experimenting with ooRexx.	164
Figure 3: LibreOffice Writer on Linux. ....	165
Figure 4: Apache OpenOffice Writer on MacOSX. ....	166
Figure 5: Apache OpenOffice Writer on Windows. ....	166
Figure 6: Windows Script Host (wsh) Popup. ....	171
Figure 7: ooRexx Homepage ( <a href="http://www.ooRexx.org">http://www.ooRexx.org</a> ). ....	174
Figure 8: ooDialog's Example "New List Controls". ....	181

## List of Tables

Table 1: Rexx Operators. ....	33
Table 2: Overview of the Rexx Built-in Functions (BIFs). ....	37
Table 3: Combining Boolean Values with "&" (AND). ....	43
Table 4: Combining Boolean Values with " " (OR). ....	44
Table 5: Combining Boolean Values with "&&" (XOR). ....	44
Table 8: Some Methods of the ooRexx Root Class "Object". ....	116
Table 9: Some Methods of the ooRexx Meta Class "Class". ....	118
Table 10: Some Methods of the ooRexx Class "Method". ....	119
Table 11: Some Methods of the ooRexx Class "Message". ....	121
Table 12: Some Methods of the ooRexx Class "Routine". ....	121
Table 13: Some Methods of the ooRexx Class "Package". ....	122
Table 14: Some Methods of the ooRexx Class "String". ....	124
Table 15: Some Methods of the ooRexx Class "Stream". ....	127
Table 16: Some Methods of the ooRexx Class "DateTime". ....	152
Table 17: Some Methods of the ooRexx Class "TimeSpan". ....	152
Table 18: Some Methods of the ooRexx Class "File". ....	153
Table 19: Some Methods of the ooRexx Class "MutableBuffer". ....	155
Table 20: Some Methods of the ooRexx Class "RexxContext". ....	156
Table 21: Some Methods of the ooRexx Class "OLEObject". ....	171

## List of Codes (Nutshell Examples)

Code 1.1-1: "Hello world!" .....	23
Code 1.3-1: Nested Block Comments. ....	27
Code 1.3-2: Line Comment. ....	28
Code 1.3-3: Rexx Clauses and Semicolons. ....	45
Code 1.3-4: Rexx Clauses (Using Comma, Dash and Semicolons). ....	46
Code 1.3-5: Rexx Program with a Null Clause. ....	46
Code 1.3-6: Rexx Clause, Spreading Over Five Lines with Whitespace. ....	48
Code 1.3-7: Normalized Clause. ....	48
Code 1.4-1: Stem Variables. ....	51
Code 1.4-2: Stem Defined with a Default Value. ....	51
Code 1.4-3: Stem Array. ....	52
Code 1.4-4: Transferring Control to a Label with SIGNAL. ....	54
Code 1.4-5: Transferring Control to a Label with CALL. ....	54

## List of Codes (Nutshell Examples)

Code 1.4-6: Invoking an Internal Routine as a Function. ....	55
Code 1.4-7: Calling an Internal Routine, that Returns a Value. ....	55
Code 1.4-8: Fetching Arguments Using the ARG() BIF. ....	55
Code 1.4-9: Changing Variables from the Caller in an Internal Routine. . .	56
Code 1.4-10: Procedure Scope - Insulating Caller's Variables. ....	57
Code 1.4-11: Procedure Scope - Exposing One Caller's Variable. ....	57
Code 1.4-12: Procedure Scope - Exposing a Stem from the Caller. ....	58
Code 1.4-13: Using a Stem Array from the REXXUTIL Function SysFileTree(). ....	59
Code 1.4-14: Message Instructions. ....	60
Code 1.4-15: Demonstrating the SAY Keyword Instruction. ....	64
Code 1.4-16: Demonstrating the IF Keyword Instruction. ....	64
Code 1.4-17: Demonstrating the IF Keyword Instruction with ELSE. ....	65
Code 1.4-18: Demonstrating the Clause Borders by Semicolons. ....	65
Code 1.4-19: An Alternative for Formatting an IF Keyword Instruction. . .	65
Code 1.4-20: Demonstrating the select Keyword Instruction. ....	66
Code 1.4-21: Demonstrating the do Keyword Instruction. ....	66
Code 1.4-22: Demonstrating the do Keyword Instruction with Repetition. .	67
Code 1.4-23: Demonstrating the do Keyword Instruction with Repetition. .	67
Code 1.4-24: Demonstrating Repetition with a Control Variable. ....	68
Code 1.4-25: Demonstrating Repetition with a Control Variable. ....	68
Code 1.4-26: Demonstrating Repetition with a Control Variable. ....	68
Code 1.4-27: Using an Explicit Label Subkeyword. ....	69
Code 1.4-28: Using the do Keyword Instruction with the WHILE Subkeyword. ....	69
Code 1.4-29: Using the do Keyword Instruction with the WHILE Subkeyword. ....	69
Code 1.4-30: Using the do Keyword Instruction with the UNTIL Subkeyword. ....	70
Code 1.4-31: Using the LEAVE and ITERATE Keyword Instructions. ....	70
Code 1.4-32: Using the FOREVER Keyword Instruction. ....	71
Code 1.4-33: Using the DO...over Keyword Instruction. ....	71
Code 1.4-34: Using the LOOP Keyword Instruction. ....	72
Code 1.4-35: Using the QUEUE, PUSH, PULL and PARSE PULL Keyword Instructions. ....	73
Code 1.4-36: Accessing the Output of an External Command via rxqueue. ....	73
Code 1.4-37: Using the PARSE Keyword Instruction. ....	74
Code 1.4-38: Using the PARSE Keyword Instruction, Version 2. ....	75
Code 1.4-39: Using the PARSE Keyword Instruction with Literal Strings. .	76
Code 1.4-40: Using the PARSE Keyword Instruction with an Expression. .	77
Code 1.4-41: Using the PARSE Keyword Instruction with Expressions. ....	77
Code 1.4-42: Using the PARSE Keyword Instruction with Positions and Lengths. ....	78
Code 1.4-43: Using the PARSE Keyword Instruction with Absolute Positions. ....	79



## List of Codes (Nutshell Examples)

Code 1.4-44: Using the PARSE Keyword Instruction with Relative Positions. ....	79
Code 1.4-45: Using the PARSE Keyword Instruction with Relative Positions. ....	80
Code 1.4-46: Using the PARSE VALUE Keyword Instruction. ....	80
Code 1.4-47: Using the PARSE VALUE Keyword Instruction with Parentheses. ....	81
Code 1.4-48: Using the PARSE ARG Keyword Instruction with Parentheses. ....	81
Code 1.4-49: Using the PARSE PULL Keyword Instruction. ....	82
Code 1.4-50: Using the PARSE SOURCE Keyword Instruction. ....	83
Code 1.4-51: Using the PARSE VERSION Keyword Instruction. ....	83
Code 1.4-52: Using the TRACE Keyword Instruction. ....	84
Code 1.4-53: Using the INTERPRET Keyword Instruction. ....	85
Code 1.4-54: Demonstrating a Command Instruction. ....	86
Code 1.4-55: Demonstrating Command Instructions to the THE Editor. ...	86
Code 2.1-1: Using the USE Keyword Instruction. ....	88
Code 2.1-2: Using the USE Keyword Instruction with the strict Subkeyword. ....	89
Code 2.2-1: Trapping a SYNTAX Condition. ....	91
Code 2.2-2: Trapping a NOVALUE Condition. ....	92
Code 2.2-3: Using the RAISE Keyword Instruction. ....	92
Code 2.2-4: Using the RAISE Keyword Instruction and Trapping the Condition. ....	93
Code 2.3-1: Demonstrating the Routine Directive. ....	94
Code 2.3-2: Routine Directive and Trapping User Defined Conditions. ....	95
Code 2.3-3: Routine Directive and Trapping User Defined Conditions. ....	96
Code 2.3-4: p1.rex calls p2.rex. ....	97
Code 2.3-5: p2.rex calls p3.rex. ....	97
Code 2.3-6: p3.rex calls p4.rex. ....	97
Code 2.3-7: p4.rex. ....	98
Code 2.3-8: p1.rex requires p2.rex. ....	99
Code 2.3-9: p2.rex requires p3.rex. ....	99
Code 2.3-10: p3.rex requires p4.rex. ....	99
Code 2.3-11: p4.rex. ....	100
Code 3-1: Mixing String Functions and Message instructions. ....	106
Code 3.1-1: The Runtime Environment and Environment Symbols. ....	109
Code 3.1-2: Sending Messages to Objects (Values, Instances). ....	110
Code 3.1-3: Using the Runtime Environment with Environment Symbols. ....	111
Code 3.1-4: Constructor and Destructor Method Routines. ....	112
Code 3.1-5: Resolving Methods Using the Class Hierarchy. ....	113
Code 3.1-6: Unknown Method Routine. ....	114
Code 3.2-1: Demonstrating Using Some Methods of .Object. ....	117
Code 3.2-2: Demonstrating Using Some Methods of .Class. ....	119
Code 3.2-3: Demonstrating Using Some Methods of .Method. ....	120
Code 3.2-4: Demonstrating Using Some Methods of .Message. ....	121

## List of Codes (Nutshell Examples)

Code 3.2-5: Demonstrating Using Some Methods of <code>.Routine</code> .	122
Code 3.2-6: Demonstrating Using Some Methods of <code>.Package</code> .	123
Code 3.2-7: Demonstrating Using Some Methods of <code>.String</code> .	125
Code 3.2-8: Using Stems the Classic Rexx Style.	125
Code 3.2-9: Intermixing Classic Rexx Stem Access and Methods of <code>.Stem</code> .	126
Code 3.2-10: Demonstrating Using Some Methods of <code>.Stem</code> .	126
Code 3.2-11: Demonstrating Using Some Methods of <code>.Stream</code> .	128
Code 3.2-12: Single Dimensioned <code>.Array</code> .	131
Code 3.2-13: Two-dimensional <code>.Array</code> .	132
Code 3.2-14: Sorting a Single Dimensioned <code>.Array</code> .	132
Code 3.2-15: Sorting a Single Dimensioned <code>.Array</code> With a Comparator.	133
Code 3.2-16: Using a <code>.List</code> to Collect and Process Objects.	134
Code 3.2-17: Using a <code>.Queue</code> to Collect and Process Objects.	135
Code 3.2-18: Using a <code>.CircularQueue</code> to Collect and Process Objects.	136
Code 3.2-19: Using a <code>.Directory</code> to Collect and Process Objects.	138
Code 3.2-20: Using the <code>.local</code> Directory to Collect and Process Objects.	139
Code 3.2-21: Using a <code>.Relation</code> to Collect and Process Objects.	141
Code 3.2-22: Using a <code>.Table</code> to Collect and Process Objects.	142
Code 3.2-23: Using a <code>.Bag</code> to Collect and Process Objects.	143
Code 3.2-24: Using a <code>.Set</code> to Collect and Process Objects.	144
Code 3.2-25: Setlike Operations with a <code>.Bag</code> and a <code>.Bag</code> .	146
Code 3.2-26: Setlike Operations with a <code>.Set</code> and a <code>.Bag</code> .	147
Code 3.2-27: Setlike Operations with a <code>.Bag</code> and a <code>.Set</code> .	148
Code 3.2-28: Demonstrating <code>.Alarm</code> .	149
Code 3.2-29: Demonstrating a Custom <code>.Comparator</code> for Sorting an <code>.Array</code> .	151
Code 3.2-30: Demonstrating <code>.DateTime</code> and <code>.TimeSpan</code> .	152
Code 3.2-31: Demonstrating <code>.File</code> .	153
Code 3.2-32: Using the <code>.output</code> Monitor.	154
Code 3.2-33: Demonstrating <code>.MutableBuffer</code> .	155
Code 3.2-34: Using <code>.context</code> (a <code>.RexxContext</code> ).	156
Code 4.1-1: Using a Java Object as If It Was an ooRexx Object.	161
Code 4.1-2: Using a Java Dialog as If It Was From an ooRexx Class.	162
Code 4.1-3: Using Java to Interact with Apache OpenOffice/LibreOffice.	163
Code 4.2-1: Using a Windows Object as If It Was an ooRexx Object.	172
Code 4.2-2: Using an Internet Explorer Object as If It Was an ooRexx Object.	173
Code 4.2-3: A Visual Basic Script (VBS) Program.	177
Code 4.2-4: An ooRexx Program Matching the Above VBS Program.	177
Code 4.2-5: A Visual Basic Application (VBA) Program.	178
Code 4.2-6: An ooRexx Program Matching the Above VBA Program.	178
Code 5.1-1: <code>.methods</code> Collecting Floating Methods.	185
Code 5.1-2: Using Messages.	186
Code 5.1-3: Using Cascading Messages.	187
Code 5.1-4: Demonstrating the Required String Value.	188
Code 5.1-5: Using a Constructor Method Routine.	189

## List of Codes (Nutshell Examples)

Code 5.1-6: Using a Destructor Method Routine. ....	190
Code 5.1-7: Using an UNKNOWN Method Routine. ....	191
Code 5.1-8: Using a STRING Method Routine. ....	192
Code 5.1-9: Using a MAKESTRING Method Routine. ....	193
Code 5.1-10: Using a MAKEARRAY Method Routine. ....	194
Code 5.1-11: Implementing a Comparison Method Routine named "=". .	196
Code 5.1-12: Implementing a Method Routine named compareTo. ....	197
Code 5.1-13: Inheriting from Orderable and Implementing the Abstract Method compareTo. ....	198
Code 5.1-14: Implementing Comparator Method Routines. ....	200
Code 5.2-1: Implementing the ADT Birthday. ....	203
Code 5.2-2: Implementing the ADT Person. ....	203
Code 5.2-3: Implementing the ADT Birthday. ....	205
Code 5.2-4: Implementing the ADT Person. ....	205
Code 5.2-5: Get and Set Methods the ooRexx Interpreter Creates. ....	206
Code 5.2-6: Example for a Method Routine. ....	207
Code 5.2-7: Implementation of the ADT Birthday. ....	208
Code 5.2-8: Implementation of the ADT Person. ....	209
Code 5.2-9: Method Routine Accesses Attribute NAME Directly. ....	210
Code 5.2-10: self and super in Method Routines. ....	211
Code 5.2-11: Using the FORWARD Keyword Instruction. ....	212
Code 5.2-12: Demonstrating the ::CONSTANT Directive. ....	213
Code 5.2-13: Using a Class Hierarchy. ....	214
Code 5.2-14: Multiple Inheritance (AmphibianVehicle class specializes the RoadVehicle class and inherits from the WaterVehicle). ....	216
Code 5.2-15: Multiple Inheritance (AmphibianVehicle class specializes the WaterVehicle class and inherits from the RoadVehicle). ....	217
Code 5.3-1: Starting Multithreading with the REPLY Keyword Instruction. ....	219
Code 5.3-2: Starting Multithreading with the .Alarm. ....	220
Code 5.3-3: Starting Multithreading with .Object's start Method. ....	221
Code 5.3-4: Starting Multithreading with .Message's start Method. ....	222
Code 5.3-5: Synchronizing Threads with the GUARD Keyword Instruction. ....	225
Code 5.3-6: Waiting for Threads. ....	227
Code 5.3-7: Synchronizing Producer with Consumer. ....	230



## **Part I**

*The REXX Language (“Everything Is a String”)*



# 1 The Rexx Language (“Everything Is a String”)

This chapter aims to introduce the Rexx language so as to enable the reader to use any (classic) Rexx interpreter like Regina, IBM host (mainframe) Rexx implementations and of course the ooRexx interpreter, which is compatible to Rexx. In order to achieve this goal any usage of specific ooRexx extensions (features) will be noted explicitly.

## 1.1 “Hello world!” in Rexx

It has become a custom to demonstrate the characteristics of a programming language by writing a small program that outputs the string *Hello world!*:

```
say "Hello world!"
```

*Code 1.1-1: “Hello world!”*

This small program demonstrates the English keyword instruction `SAY` (which indicates its purpose) and is followed by a literal string enclosed in double-quotes. Please note that Rexx is “caseless”, so it does not matter in which case the `SAY` keyword instruction was spelled, hence all the following spellings are fine and are taken to mean the `SAY` keyword instruction: `SAY`, `Say`, `saY`, `SaY`, `sAY`, `SAY`, `say`.

Rexx programs are made of plain text and get saved in plain (ASCII) text files with a file extension of `.rex`. If the above Rexx program was saved in a file named `hello.rex`, then this Rexx program can be executed by entering the command `rexX hello.rex` on the command line. After pressing the return key ↵ the Rexx interpreter loads the Rexx program from the text file `hello.rex` and executes it line by line. The `SAY` keyword instruction will output the given literal string, which will show the following result in the command line window:

```
Hello world!
```

It is also possible to run Rexx programs via the graphical user interface of the operating system by double-clicking the file directly in the folder. Alternatively, one may hover the mouse over the Rexx file, press the right mouse-button to open the context menu and choose `Run` from the menu. In this case the operating system will open a command line window and have

## 1.1 “Hello world!” in Rexx

the Rexx interpreter execute the Rexx program. Upon termination of the Rexx program, the command line window will be closed automatically as well by the operating system. On modern computers this will be so fast that the user might not notice these steps, apart from maybe a brief flicker.

- ☞ Hint: to keep the command line window open in this use case, add the statement `PARSE PULL xyz` as the last instruction in your Rexx program. This instruction will wait for user input via the keyboard and assign the input to the indicated variable once the user presses the return key ↵. Only thereafter will the Rexx program terminate and the operating system would close the command line window. Our program may therefore be rewritten like this:

```
SAY "Hello world!"  
PARSE PULL xyz
```

Running the above Rexx program will output the string `Hello world!` and then process the `PARSE PULL` keyword instruction which waits for the user to enter something via the keyboard. After pressing the return key ↵ the `PARSE PULL` instruction will assign the keyboard input to the variable `xyz` and as there are no more instructions left in the program the Rexx interpreter will stop executing and return control to the operating system.

## 1.2 **Fundamental Language Concepts**

The author of the Rexx language, Michael F. Cowlshaw, discusses the design principles of the REXX language in his book “The REXX Language”:<sup>3</sup>

- *Readability*: the structure of the syntax and the names of the keyword instructions should be easily readable. Although Rexx does not distinguish between lower- and uppercase, one can use case to make programs more legible.
- *Natural data typing*: unlike many languages, Rexx is not strongly typed. Everything is a string and its meaning depends only on its usage. For example, arithmetic operations would cause the interpreter to check whether the string operands contained valid numbers to carry out the desired arithmetic and would raise a runtime error if not.

<sup>3</sup> The following list of language concepts is taken from <http://speleotrove.com/rexxhist/TRL-background.pdf> (as of March 2013), made available to the public by Mike F. Cowlshaw .



## 1 The Rexx Language (“Everything Is a String”)

- *Emphasis on symbolic manipulation:* as everything in Rexx is a string there is a rich set of string manipulation operators and functions. One of the most common string operations is concatenation for which Rexx supplies two different means: concatenating strings by intervening white space (space or tabulator character) which becomes part of the concatenated string or by an explicit concatenation operator (two vertical bars, `||`) that concatenates two strings without an intervening blank.
- *Dynamic scoping:* Rexx scoping adheres to the execution sequence of Rexx clauses as defined by the programmer.
- *Nothing to declare:* there is no mechanism for defining variables. If a variable is needed one can use a variable symbol.
- *System independence:* the language is defined in a manner which makes it independent of the hardware and system software (although it is possible to interact with the environment).
- *Limited span syntactic units:* syntactic units in Rexx are usually `clauses`, which span a single line only. If an error is detected or a program is traced, then these units are used to give the programmer good diagnostics.
- *Dealing with reality:* although consistency has been a major design goal, if in real use it creates unexpected side effects, then these were addressed in the language's design as was the case with the `TRACE` keyword instruction.
- *Be adaptable:* Rexx does not reserve any keywords, such that the language can be adapted without unexpected side effects should new keyword instructions be needed.
- *Keep the language small:* features were only added, if they were of use for a significant number of users. A small language can be learned fast and memorized for a long time.
- *No defined size or shape limits:* the Rexx language does not define any limits on the size or shape of its `TOKENS`<sup>4</sup> or data.<sup>5</sup>

---

<sup>4</sup> A `TOKEN` is the unit of low-level syntax from which one constructs Rexx clauses. Cf. subsection ??, p. 45 below.

<sup>5</sup> Different implementations of Rexx interpreters may have implementation dependent limits.

## 1.2 Fundamental Language Concepts

All of these concepts should foster the definition of a “human centric” language.

### 1.3 **Building Blocks and Definitions**

This section defines the allowable characters to create Rexx programs and the building blocks of a Rexx program.

#### 1.3.1 **Characters Allowed in a Rexx Program**

A Rexx program may consist of the following characters:

- alphabetical lowercase characters: **abcdefghijklmnopqrstuvwxyz**
- alphabetical uppercase characters: **ABCDEFGHIJKLMNOPQRSTUVWXYZ**
- digit characters: **0123456789**
- the following characters: **!** (exclamation mark), **?** (question mark), **\_** (underscore)
- the following *special characters* for defining clauses and expressions (each of the following characters is regarded to be a *TOKEN* that can be distinguished from any other *TOKEN* in a clause): **;** (semicolon), **:** (colon), **.** (dot), **+** (plus), **-** (minus), **\*** (asterisk), **/** (slash), **,** (comma), **=** (equal sign), **|** (vertical bar), **\** (backslash), **~** (“not” character), **(** (open parenthesis), **)** (close parenthesis)
- in addition, the following *special characters* in **ooRexx** (each of the following characters is regarded to be a *TOKEN* that can be distinguished from any other *TOKEN* in a clause): **~** (tilde), **[** (square open parenthesis), **]** (square close parenthesis)
- whitespace (non-visible) characters: space/blank character () and tabulator character (**→**)

In addition literal (quoted) strings and comments may contain any character.

The allowable characters can be used for different purposes when writing Rexx programs.